Submitted to ICML-98

Reinforcement Learning with Policy Constraints

Sebastian Thrun **Jamieson E. Schulte**
Computer Science Department
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213-3891

Abstract

This paper addresses the problem of knowledge transfer in lifelong reinforcement learning. It proposes an algorithm which learns policy constraints, i.e., rules that characterize action selection in entire families of reinforcement learning tasks. Once learned, policy constraints are used to bias learning in future, similar reinforcement learning tasks. The appropriateness of the algorithm is demonstrated in two domains: A grid world domain and a (more challenging) light control problem for commercial office space.

1 Introduction

In recent years, the field of reinforcement learning [2, 26, 33] has received significant attention in the AI community. Recent research has led to a much improved theoretical understanding of the capabilities and limitations of reinforcement learning [1, 2, 4, 11, 7, 10]. Various researchers have successfully applied reinforcement learning techniques to challenging real-world problems [5, 25, 29, 34].

The vast majority of the work, however, has focused on situations where a reinforcement learner faces a single task. In this paper we consider the problem where a reinforcement learner encounters a multitude of learning tasks [30, 31]. For example, a long-living mobile robot might have to perform many different tasks over its lifetime, and it might have to operate in many different environments. In such situations, there is often an opportunity to transfer knowledge among the different tasks/environments, so that the learner can avoid having to learn from scratch. For example, a mobile robot might learn that avoiding collisions with obstacles is a generically useful thing to do, independent of the specific environment or task at hand. Re-using such previously learned knowledge has frequently been found to increase the efficiency of learning in new tasks/environments (see various chapters in [31]).

This paper proposes an approach that learns and transfers *policy constraints* among multiple reinforcement learning tasks. At the bottom level, our approach employs Q-learning [33]to learn control policies for individual tasks. On top of that, our approach employs a decision tree learning algorithm [21] for learning *policy constraints*. Policy constraints specify what action to perform or to avoid in specific situations, conditioned on state features. They are encoded through propositional rules of the type *"If $P(s)$ do/do not a,"* where $P(s)$ denotes a propositional condition on the state s, and a denotes an action. When the Q-learner is placed in a new environment or is asked to perform a new task, previously learned policy constraints are used to guide learning.

To demonstrate the appropriateness of the approach, this paper reports experimental results obtained in two different domains: A grid world domain, and a more challenging building control domain. Grid world has been chosen since it permits systematic experimentation, necessary to evaluate the relative benefits of the approach.

The building domain, which in fact has provided much of the motivation for the work described here, involves the control of light redirection louvers in an existing commercial office space. The testbed is the "Robert L. Preger Intelligent Workplace," an office building constructed by CMU's Architecture Department for research on innovative building design and operation. To balance natural and artificial illuminance, the building is equipped with motorized light-redirection louvers made of semi-reflective, tinted glass, which can reflect away incoming light. The opening angle of the louvers can be controlled by a computer. There exists already a control algorithm, which has been carefully tuned by hand to meet the properties of the specific building. This algorithm is used as a benchmark.

The building control problem casts itself nicely as a lifelong learning problem. Various aspects of the problem are subject to change, such as: the occupants, their individual preferences, their usage of the building, the energy costs, and the physics of the building itself (*e.g.*, location and color of walls). Initial experiments in the building domain suggest that the controller learned by our approach outperforms the commercial controller. This paper also provides examples of useful policy constraints obtained in the building domain.

Figure 1: The "Intelligent Workplace" (IW) is the top floor of the building shown here. The is currently being equipped with hundreds of sensors and a variety of computer-controllable devices. The light redirection louvers are already under computer control.

2 Q-Learning

Our approach is based on Watkins's Q-learning [33]. Since the basic Q-learning algorithm is sufficiently well-known, we will only sketch it here and refer the interested reader to the rich literature on this topic (see *e.g.*, [4, 13, 28]).

Q-learning is an algorithm for learning mappings from states to actions (called: *control policies*) that, if applied for action selection, maximize an externally provided payoff measure. Let s denote a state, a an action, and $r(s, a)$ an (initially unknown) function that specifies the *immediate payoff* obtained for executing action a in state s. A *control policy* is a mapping from states to actions $\pi : s \longrightarrow a$. The objective of Q-learning is to learn a control policy that, if applied for action selection, maximizes the sum of future payoff the Q-learner is expected to receive:

$$E[\sum_{t=0}^{\infty} \gamma^t r(s^{(t)}, a^{(t)}) \mid s^{(0)} = s]. \tag{1}$$

Here s denotes the current state, $s^{(t)}$ denotes the state at time t, $E[\ ]$ denotes the expected value, and γ is a commonly used *discount factor* ($0 \leq \gamma \leq 1$) which discounts payoffs exponentially with time.

Q-learning learns π indirectly, by learning a value function $Q(s,a)$. After learning, $Q(s,a)$ measures the *goodness* of action a in state s: the larger $Q(s,a)$, the better that action. The function Q is learned from observations. More specifically, suppose the learner just executed an action a at a state s. In response, the environment's state changed to s' and the agent received an immediate payoff $r(s,a)$. This observation is used to update the function Q:

$$Q(s,a) \quad \longleftarrow \quad \alpha \left(r(s,a) + \gamma \operatorname*{argmax}_{a'} Q(s',a') \right) + (1-\alpha)\ Q(s,a) \tag{2}$$

where $\alpha > 0$ is a learning rate that must be appropriately decreased over time. A common scheme for action selection during learning is *Boltzmann exploration* [2], which picks actions stochastically, but under consideration of their current Q-value.

Under appropriate conditions [11], repetitive application of this update rule is guaranteed to converge to a value function that is *optimal*, in the sense that it maximizes (1). More specifically, the greedy policy

$$\pi_{\text{greedy}}(s) \quad := \quad \operatorname*{argmax}_{a} Q(s,a) \tag{3}$$

can be shown to be an optimal control policy [3, 28]. Most existing convergence results rely on a strict set of assumptions, including: the environment must be Markov, the Q-function must be represented by a lookup table, and each state-action pair must be visited infinitely often during learning. In practice, however, Q-learning has been applied with remarkable success under a much broader range of conditions [5, 25, 29, 34].

3 Policy Constraints

Policy constraints are conditional *if-then* rules that prescribe for specific situations what action to take, or which one to avoid. They are extracted from one or more control policies, and are used to initialize the Q-function when learning new policies for similar tasks.

3.1 Extraction

Our approach extracts policy constraints in four steps: (1) learning a decision tree that represents the policy, (2) converting the tree into rules, (3) simplifying the resulting rules, and (4) post-pruning them. This approach is commonly used in the context of supervised learning [17].

1. **Decision Tree Learning.** First, the policy π is converted into a decision tree, using ID3, a standard decision tree learning algorithm [21]. Each branch in the tree is a test on a specific state feature attribute, and each leaf of the tree is a unique action. When growing the tree, the standard information gain criterion is applied to access the utility of each state feature. If the tree is generated from a single policy (single task), it is always possible to generate a tree that describes π accurately. If, however, a tree is generated from multiple control policies, it might happen that a leaf corresponds to multiple actions.

2. **Conversion to rules.** Each leaf node in the decision tree is converted into a set of rules. There are two types of rules, *positive* and *negative* rules. If a leaf is labeled by exactly one action, a rule of the type

   If $P(s)$ *do* a

   is generated. Such rules are called *positive*. Here $P(s)$ is the conjunct of all tests along the path between the root and the leaf, and a is the corresponding action. If the leaf contains more than one action, no positive rule is generated.

   For all actions a' that are *not* listed in the leaf, rules of the following type are generated:

   If $P(s)$ *do not* a'.

   Such rules are called *negative rules*. Each leaf in the tree generates at most as many rules as there are actions; it might generate none.

3. **Simplifying the rules.** The rule set is simplified in four steps, each of which retains the same (full) coverage of the original rule set. First, each rule is simplified by removing all antecedents (preconditions) that do not change the consistency between the rule and the data. Second, rules that are specializations of other rules are removed. Third, rules that follow from other rules by the fact that the agent has to chose exactly one action are removed—this step often removes negative rules rendered unnecessary by a positive rule. Fourth, identical copies of rules are removed. None of these steps alters the consistency of the rule set with the data; however, the simplification can greatly reduce the complexity and the total number of the rules.

4. **Pruning.** Finally, rules are removed that appear to overfit the specific control policies. In our current approach, pruning is realized by two simple criteria:

(1) Rules are removed that are overly specific, *i.e.*, which possess more antecedents than a predefined threshold. (2) Rules that are not supported by sufficiently many states (data) are also removed. In both cases, the threshold is set by hand. Positive and negative rules may be pruned using different thresholds.

The result of this procedure is a set of policy constraints that contains two types of rules: positive and negative. Positive rules prescribe specific actions which a learner should perform in specific situations. Negative rules describe circumstances under which specific actions should be avoided.

Usually, policy constraints are *incomplete*, that is, they apply only to a subset of all states. When transferred to a different environment, policy constraints can also be *inconsistent*, in the sense that there might exist state action pairs which are covered by a positive *and* a negative policy constraint.

3.2 Biasing Q-Learning

When learning a control policy for a new task, previously learned policy constraints are used to bias Q-learning, if available. Our approach initializes the Q table using the following, obvious scheme:

1. If a state action pair $\langle s, a\rangle$ is covered by one or more negative rules, but not by a positive rule, $Q(s, a)$ is initialized with a large negative number.

2. If a state action pair $\langle s, a\rangle$ is *not* covered by a positive rule, but there exists a positive rule that prescribes a different action a' (with $a' \neq a$) for state s, $Q(s, a)$ is initialized with a large negative number.

3. Otherwise, $Q(s, a)$ is initialized with a default value (which is 0 in all our experiments).

This procedure basically initializes the Q-table in accordance to the policy constraints. It treats possible inconsistencies in the policy constraints just as if there were no rule. In the absence of any policy constraints, Q is initialized with default values, and the approach degrades to conventional Q-learning.

The initialization scheme has two effects on Q-learning. First, it biases exploration. Actions that are good (bad) according to the policy constraints are selected more (less) frequently during learning. Second, it primes the control policy. A policy generated from the initial value function, without any learning, would automatically obey all policy constraints, as long as they are consistent. As learning progresses,

```
#####################  #####################  #####################  #####################  #####################
#...#...............#  #G..#...#...#...#...#  #...#...#...#...#...#  #...................#  #...........#..#....#
#...#...####....#...#  ##..#.#.#.#.#.#.#.#.#  ##..#.#.#.#.#.#.#.#.#  #...................#  #..#######..#..#....#
#.###..#.....#..#####  #.#...#...#...#...#.#  #.#...#...#...#...#.#  #...................#  #..#.....#..#..####.#
#.....#.......#.....#  #..#######.###.#.##.#  #..#######.###.#.##.#  #......#####........#  #..#.....#..#.......#
#....#...###...#....#  #.....#.........#...#  #.....#.........#...#  #......#............#  #..#..#..#..######..#
#.........G.........#  #.##..#..#####..#.###  #.##..#..#####..#.###  #......#............#  #.....#..#.......#..#
#....#...###...#....#  #...#.#.#.......#...#  #...#.#.#.......#...#  #......#............#  #.#...#..######..#..#
#.....#.......#.....#  ###.#...#..###.#.##.#  ###.#...#..###.#.##.#  #............G......#  #...#.#.......#..#..#
#####..#.....#..###.#  #....####.#...#...#.#  #....####.#...#...#.#  #...................#  #.#...######..#..#..#
#...#....####...#...#  #.###...#.#.#.#.#.#.#  #.###...#.#.#.#.#.#.#  #...................#  #...#....G....#..#..#
#...............#...#  #.....#.#...#...#...#  #.....#G#...#...#...#  #...................#  #.............#.....#
#####################  #####################  #####################  #####################  #####################
```

Figure 2: The grid worlds used in our experiments.

however, the initial bias provided by the policy constraints can be overridden by data.

It is important to notice that policy constraints impose only *soft constraints* on Q-learning. Q-learning can still converge to control policies that violate these constraints. If policy constraints were *hard constraints*, meaning that they could not be overridden, the algorithm would be bound to fail if a previously learned policy constraint was incorrect. The ability to recover from erroneous policy constraints is therefore essential.

4 Experimental Results

4.1 Grid World

The first sets of results were obtained using a family of grid worlds shown in Figure 2. Here the symbol "#" indicates the presence of an obstacle, "." indicates a free (traversable) position, and "G" marks the location of the goal. Initially, the learner is placed at a random position. It can chose among four actions: *go south, west, north, or east*. The learner can perceive the relative displacement of the goal state along the two coordinate axes (x and y). It can also sense the labels (*occupied*, *free*, and *goal*) of the eight surrounding states (*cf.* [14, 20]). Thus, its state representation possesses ten (partially redundant) features: two relative goal distances, and eight state labels. The effects of actions are stochastic. With probability p, actions move the agent into a random direction. If the agent attempts to move into an obstacle, its position does not change and it receives a -1 penalty instead. When it reaches the goal, it receives a $+100$ payoff and gets restarted in a random position. The learning task is to devise a control policy that safely carries the learner to the goal on a shortest past. In all our experiments, the first four grid worlds in Figure 2 were used for generating policy constraints; the fifth was used to

Rule set C_1: Rules extracted from the first grid world				
Name	**Rule**		**Type**	**Coverage**
r_1	goal(east)	$\rightarrow$ go(east)	pos.	1
r_2	goal(west)	$\rightarrow$ go(west)	pos.	1
r_4	occupied(east)	$\rightarrow$ $\neg$go(east)	neg.	35
r_5	occupied(west)	$\rightarrow$ $\neg$go(west)	neg.	35
r_6	occupied(north)	$\rightarrow$ $\neg$go(north)	neg.	56
r_7	occupied(south)	$\rightarrow$ $\neg$go(south)	neg.	56

Runner-ups (these are pruned out)				
Name	**Rule**		**Type**	**Coverage**
r_8	free(north) $\wedge$ goal_dist(y,+2)	$\rightarrow$ go(north)	pos.	12
r_9	free(north) $\wedge$ goal_dist(y,+3)	$\rightarrow$ go(north)	pos.	8
r_{10}	occupied(north_west) $\wedge$ free(east) $\wedge$ occupied(north_east)	$\rightarrow$ $\neg$go(north)	neg.	28
r_{11}	free(south_west) $\wedge$ free(east) $\wedge$ occupied(north_east)	$\rightarrow$ $\neg$go(north)	neg.	28

Table 1: Rule set C_1, extracted from the first grid world, along with some of the closest runner-ups.

evaluate their effects in Q-learning. Positive rules are pruned if they possess more than one antecedent. Negative rules are pruned if they are supported by less than $30N$ states, where N is the total number of tasks. Notice that in each task, two antecedents (the relative goal distances) are sufficient to uniquely identify the state, hence the strict pruning rules.

Tables 1 and 2 show rules extracted from the first grid world (rule set C_1), and from the first four grid worlds (rule set $C_{1\text{-}4}$), respectively. Here "coverage" refers to the number of states covered by the respective rule. Both tables also show the closest runner-ups which, however, are pruned away. These rules can be grouped into two categories: *Collision avoidance rules*, and *goal seeking rules*. Collision avoidance rules are negative rules (*"don't run into an obstacles"*), whereas goal seeking rules are positive rules (*"go to the goal if you can see it"*). Both tables show the rules *after* pruning the rule set. Before pruning, there are 143 positive and 55 negative rules for the first grid world, and 474 positive and 342 negative rules for the second. Almost all of those rules are only supported by a single state, *i.e.*, they basically specify what to do in that specific state. Our pruning mechanism retains

Rule set $C_{1\text{-}4}$: Rules extracted from the first four grid worlds				
Name	Rule		Type	Coverage
r_1	goal(north)	$\rightarrow$ go(north)	pos.	1
r_2	goal(south_west)	$\rightarrow$ go(south)	pos.	1
r_3	goal(east)	$\rightarrow$ go(east)	pos.	2
r_4	goal(south)	$\rightarrow$ go(south)	pos.	2
r_5	goal(north_west)	$\rightarrow$ go(north)	pos.	2
r_6	goal(south_east)	$\rightarrow$ go(east)	pos.	2
r_7	goal(west)	$\rightarrow$ go(west)	pos.	3
r_8	occupied(east)	$\rightarrow \neg$go(east)	neg.	172
r_9	occupied(west)	$\rightarrow \neg$go(west)	neg.	173
r_{10}	occupied(north)	$\rightarrow \neg$go(north)	neg.	204
r_{11}	occupied(south)	$\rightarrow \neg$go(south)	neg.	205

Runner-ups (these are pruned out)				
Name	Rule		Type	Coverage
r_{12}	goal_dist(x,-6) $\wedge$ goal_dist(y,+3)	$\rightarrow$ go(north)	pos.	3
r_{13}	goal_dist(x,+1) $\wedge$ goal_dist(y,-4)	$\rightarrow$ go(south)	pos.	3
r_{14}	free(north_west) $\wedge$ free(north) $\wedge$ goal_dist(y,+1)	$\rightarrow$ go(north)	pos.	23
r_{15}	free(west) $\wedge$ occupied(south_west) $\wedge$ free(north_east) $\wedge$ free(east)	$\rightarrow \neg$go(south)	neg.	69

Table 2: Rule set $C_{1\text{-}4}$, extracted from the first four grid worlds, along with some of the closest runner-ups.

only the most general/compact rules.

The performance result obtained for the fifth grid world shown in Figure 2 are summarized in Table 3 and Figure 3. Table 3 shows statistics for the time it takes to find the goal for the first time during learning. It also gives results for the number of collisions encountered during this search. The percentages shown in Table 3 describe the relative improvement induced by the policy constraints. As can be seen there, policy constraints reduce the search time by 22% or more, while the number of collisions is reduced by 74% or more. For example, with a noise level of $p = 10\%$, previously learned policy constraints reduce the number of collisions by as much as 89.6%. Whereas the basic Q-learning approach suffers, on average, 1058 collisions, our approach reduces this to 109, most of which are caused by the stochasticity in the environment. Each entry in Table 3 has been averaged over

noise level	rule set	steps	%-improv.	collisions	%-improv.
0%	–	3,642 ±324		1,091 ±95.9	
	C_1	2,364 ±232	35.1%	32 ±5.37	97.0%
	$C_{1\text{-}4}$	2,335 ±230	35.9%	31 ±5.35	97.1%
5%	–	3,655 ±326		1,107 ±97.9	
	C_1	2,503 ±235	31.5%	70 ±9.52	93.6%
	$C_{1\text{-}4}$	2,482 ±234	32.1%	70 ±9.51	93.7%
10%	–	3,470 ±320		1,058 ±97.0	
	C_1	2,557 ±238	26.3%	111 ±12.3	89.4%
	$C_{1\text{-}4}$	2,504 ±235	27.8%	109 ±12.1	89.6%
15%	–	3,683 ±336		1,131 ±102	
	C_1	2,665 ±246	27.6%	155 ±15.8	86.2%
	$C_{1\text{-}4}$	2,652 ±245	28.0%	154 ±15.8	86.3%
20%	–	3,650 ±325		1,127 ±100	
	C_1	2,553 ±247	30.0%	190 ±20.4	83.2%
	$C_{1\text{-}4}$	2,501 ±242	31.5%	186 ±20.1	83.5%
25%	–	3,577 ±326		1,113 ±101	
	C_1	2,662 ±243	25.6%	236 ±23.1	78.8%
	$C_{1\text{-}4}$	2,646 ±243	26.0%	234 ±23.1	78.9%
30%	–	3,461 ±328		1,083 ±102	
	C_1	2,699 ±243	22.0%	281 ±26.9	74.0%
	$C_{1\text{-}4}$	2,683 ±243	22.5%	280 ±26.9	74.1%

Table 3: Summary statistics for finding the goal in the grid world domain, for different noise levels, different rule sets, and broken down into number of steps needed for finding the goal, and number of collisions encountered. The field "%-improv." shows the relative improvement caused by the policy constraints. Each number is averaged over 1,000 independend experiments. 95% confidence intervals are also shown.

1,000 experiments. All improvements are statistically significant at the 95% level.

Figure 3 plots the average number of collisions during learning, when no rules are used (thin curve), the rule set C_1 is used (gray curve), and the rule set $C_{1\text{-}4}$ is used (thick curve). Note the logarithmic scale of the vertical axis. When measuring these values, exploration was turned off. The noise level in these experiments was p=10%. As is easy to be seen there, the rule-based approach yields much better results. For example, the average number of collisions of plain Q-learning after 250,000 learning steps is 7.89±1.14, whereas our approach with rule set $C_{1\text{-}4}$ leads, on average, to only 1.14±0.09 collisions—a reduction of 85.6%. Even after 1,000,000 learning steps, after which we terminated each experiment, plain Q-learning has not even reached the level of performance that our approach reaches after 200,000 learning steps. Eventually, however, both algorithms converge to the same, optimal policy. All these results are significant at the 95% confidence level.

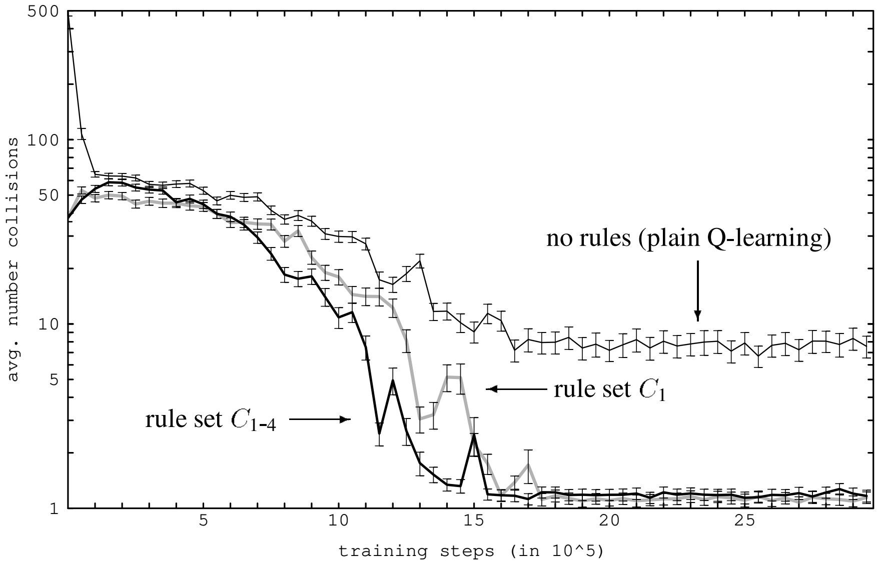

Figure 3: Learning curves. Shown here is the average number of collisions (with exploration switched off) as a function of learning steps, along with 95% confidence intervals. The results have been averaged over 10 independent runs and all possible starting positions.

4.2 Light Control in the "Intelligent Workplace"

The second testbed for our algorithm is the "Intelligent Workplace" (IW), the building shown in Figure 1. The task is to control the light-redirection louvers so as to maximize a combination of criteria:

1. **Set point.** The illuminance inside the office should be as closely as possible to the set point of approximately 1,000 Lux. The penalty function for deviating from the set point is non-linear.

2. **Spatial Uniformity.** The light level (illuminance) inside an office should be uniform as possible, so that the eyes of office workers do not have to adjust when turning the head around.

3. **Temporal Uniformity.** Over time, the illuminance should change as slowly as possible.

4. **Minimizing distraction.** The overall motion of the louvers is to be minimized, since louver motion may distract office workers.

Negative rules:

1. Don't open the louver if it is already completely open (21).
2. Don't close the louver if the louver is completely closed (21).
3. Don't close the louver if the louver is mostly closed (21).
4. Don't close the louver if it is very dark outside (16).
5. Don't close the louver if it is much darker inside than is desired (32).

Positive rules:

5. Open the louver if it is closed *and* it's much darker inside than is desired (7).
6. Open the louver if the louver is closed *and* the current illuminance is close to the set point (6).
7. No motion if the louver is mostly open *and* the outdoor brightness is moderately high (7).
8. No motion between 8am and 9am if the louver is mostly open (5).

Table 4: Rules extracted for the control of the light redirection louver in the IW. The number in parentheses indicate the number of data points that support this rule.

There are various factors that make the control of the louvers difficult. First, the exact sunlight depends on various quantities (time of day, weather, season, position of the sun), some of which are not observable (such as the weather). Second, the functional relation between the angle of the light redirection louvers and the indoor illuminance is highly non-linear and depends on various other quantities, such as the sun position and the light intensity. Third, there does not exist an sufficiently accurate model (simulator) of the building.

The input to the reinforcement learning algorithms were (1) light measurements from an array of five light sensors, mounted at the building's envelope, (2) measurements from an illuminance sensor mounted at the building's roof, (3) the current louver angle, and (4) the time of day. The reinforcement learner was given three louver motion commands (actions):*close*, *open*, and *don't move*. Data was collected over a six week period, starting Dec 24, 1997. In addition to the various sensor measurements, we also recorded the decision of the commercial controller. The first four weeks of data were used for training, and the last two weeks for evaluation.

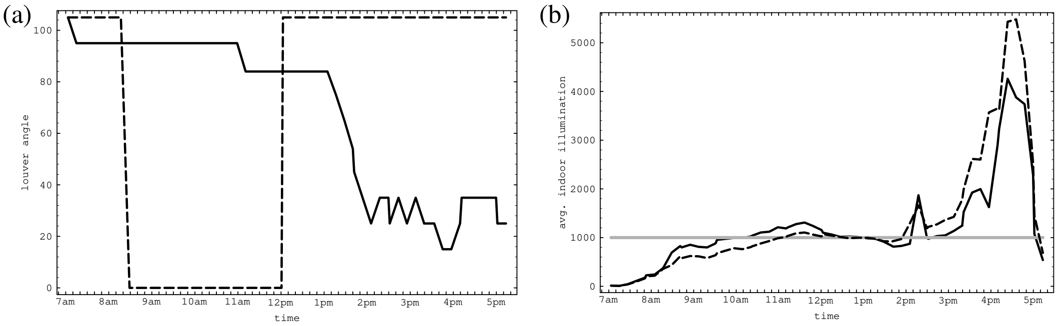

Figure 4: Example day in the Intelligent Workplace (Feb 15, 1998). (a) Louver angle. in degrees (b) Average indoor illuminance in Lux. The dashed lines correspond to the commercial controller, and the solid lines correspond to the controller learned using the approach described in this paper. The gray horizontal line in (b) indicates the desired illuminance. The reinforcement-based controller compensates much better the burst of sunlight in the afternoon.

Figure 4 shows the policy constraints extracted from the learned control policy. In contrast to the constraints extracted in the grid world domain, which characterize only immediate action effects, these constraints include several rules with strategic character. All these rules are highly meaningful, and match our intuition. The first two rules prohibit moving the louvers into their extreme positions, which is penalized due to possible hardware damages. The third constraint is interesting: it outrules a whole range of louver angles which, due to the non-monotonic response characteristics of the louvers, is not needed for optimal control. Rule 4 and 5 are obviously correct. The correctness of the last three rules is less obvious, but they nevertheless accurately describe what to do in the domain at hand.

Figure 4 illustrates the learned controller on a sunny day (see figure caption). In particular, it compares its performance to that of an existing controller, which has been designed by a company specializing in building control. From all we can tell, the existing controller behaves rather erratic on bright, sunny days. It closes the louvers between 8:30am and 12:15pm, but it ignores the fact that direct sunlight enters the building in the late afternoon. Such sunlight causes substantial glare on the computer screens inside the building, and is thus to be avoided with high priority. The controller trained with our approach does exactly the right thing: It slowly closes the louver during the day, in anticipation of the burst of sunlight in the afternoon (a unique feature of reinforcement learning with delayed reward!). As the sunlight is most intense, it carefully moves the louver to an angle where it is most effective (approx, 20 to 30 degrees), and keeps fine-tuning its precise opening angle. The effect on the indoor illuminance is quite substantial. As Figure 4b illustrates, the indoor illuminance is approximately 1000 Lux smaller at peak time than with the existing controller, reducing the glare by a clearly noticeable amount. In addition, the smoothness of the louver motion, shown in Figure 4a, reduces the distraction of the office workers.

A quantitative comparison, carried out using the data collected thus far, indicated that our approach leads to a 43.4% improvement over the commercial controller. This improvement is measured using the weighted combination of the various target functions described above. While much of the improvement can be attributed to the erratic behavior of the existing control algorithm, the results presented here demonstrate that it is feasible to design controllers that can automatically adapt to the specific building, thereby reducing the installation costs while maximizing their effectiveness. Future experiment will investigate more systematically the utility of the learned policy constraints in lifelong building control.

5 Related Work

The transfer of knowledge across multiple reinforcement learning tasks has previously been studied by several researchers. A popular approach is to acquire *action models*, which describe the state transition probabilities of individual actions [8, 18, 27, 30]. Action models can be employed to generate synthetic training data [27], or to explain observations [8, 30]. Such approaches work well if the state space dynamics stay the same across multiple tasks. In this paper, however, we are interested in cases where even the state space dynamics may change across different tasks. In such situations, it is unclear how to reuse previously learned action models.

Other researchers have proposed methods that transfer aspects of the value function. Ring, for example, proposed a neural network architecture that learns Q-functions through a series of mazes with increasing complexity [23]. Such an approach is applicable if the value functions are sufficiently similar in multiple tasks; which is not the case in the domains studied here.

Our approach to decision tree learning, rule conversion and post-pruning are standard methodology in the field of supervised learning (see *e.g.*, [17, 22]). The idea of learning decision trees in the context of reinforcement learning is also not new. Others, such as Moore and McCallum [16, 19], have proposed methods that use tree representations to partition the state space, just like ours. The key difference to our approach lies in the functions that the trees represent. In previous work, trees represent value functions, whereas our approach uses trees to represent the control policy. Since the control policy is a function of the value function (and *not* vice versa), it seems plausible that control policies can be represented by more compact trees than value functions. For example, in the grid world domain, a value function tree will never learn rules like the collision avoidance rules described above, since the *value* of a collision depends on the distance of an obstacle to the goal.

Our approach is also related to the rich literature on abstraction in reinforcement learning. A popular idea here is to decompose the policy into small sets of *partial policies* [6, 12, 24, 32]. If the decomposition matches the structure of the reinforcement learning tasks, such approaches can conceivable have the same effect, when applied to multiple reinforcement learning tasks. The question of *learning* the right decomposition is currently subject to active research.

6 Conclusion

This paper proposed a method for learning policy constraints from families of reinforcement learning tasks, and using those as a bias in other, related reinforcement learning tasks. Policy constraints are expressed by rules, which are learned from control policies using a decision tree learner. These rules specify what actions to chose in certain situations, and which ones to avoid. When learning a new task, they are used to prime the initial value function, which changes the behavior of the learner. Our approach was found to consistently outperform conventional Q-learning by a large margin and along various dimensions. Experiments in the building domain suggest that our approach outperforms an existing commercial controller by more than 40%.

Our approach makes several implicit assumptions concerning the nature of the domain. Obviously, our approach can only succeed if the different learning tasks share similar policy constraints. If, for example, different tasks possess vastly different policy constraints, reusing previously learned policy constraints can decelerate learning, although our algorithm will eventually converge to an optimal control policy. We believe that the assumption made here is not unreasonable, as many task families studied in AI share patterns in policy space. For example, avoiding collisions is generically important in mobile robotics, for a large number of different tasks and environments.

Another worst case arises if for determining whether to execute or avoid an action, *all* state features have to be considered. Such situations may arise, for example, if there exists no featured state representation (*i.e.*, states are identified by a single number), or if the state labels are assigned at random. In such situations, our approach would most likely not generate any policy constraints, and therefore degrade to conventional Q-learning.

Our algorithm is targeted towards domains where certain actions can be rejected or accepted after inspecting only a small number of state features. We believe that this is the case for many problems studied in AI. Sensor-based collision avoidance with mobile robots, light control in the building domain, and many of the popular examples studied in the field of *reactive AI* are examples of such domains. We conjecture that policy constraints can describe a wide variety of generic behavioral patterns that, once found, may provide useful guidance for future learning.

The algorithm proposed here can be extended in several directions. In our current implementation, policy constraints are only learned after convergence of Q-learning. In principle, they could also be learned *during* Q-learning, thereby providing a mechanism for transferring knowledge within a single task. Such an approach would generalize across states in Q-learning, and thus would relate to the rich body on function approximation in reinforcement learning (see *e.g.*, [1, 4, 9]).

A second limitation arises from the fact that our current approach has only been applied to table look-up representations, thereby inheriting their intrinsic scaling limitations. Table look-up representations were chosen since they facilitate systematic evaluations. An important extension would be to integrate generalizing function approximators into our approach. While the extraction of policy constraints appears to be straightforward, we currently lack an effective method for incorporating them into subsequent learning. Such an extension, however, is likely to yield improved results in complex real-world domains such as the building control domain investigated here.

Acknowledgment

The authors thank David Andre, Sila Berkol, Seongju Chan, Volker Hartkopf, Ardeshir Mahdavi, Andrew McCallum, and Bryan Singer for inspiring discussions and support during the course of this research. Partial financial support through the Advanced Building Systems Integration Consortium (ABSIC) and the National Science Foundation is gratefully acknowledged.

References

[1] L.C. Baird. Residual algorithms: Reinforcement learning with function approximation. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, San Francisco, CA, 1995. Morgan Kaufman Publishers.

[2] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[3] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[4] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[5] R.H. Crites and A.G. Barto. Improving elevator performance using reinforcement learning. In D. Touretzky, M. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, Cambridge, MA, 1996. MIT Press.

[6] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann.

[7] P. Dayan and T. J. Sejnowski. TD(λ) converges with probability 1. February 1993.

[8] T. G. Dietterich and N. S. Flann. Explanation-based learning and reinforcement learning: A unified view. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[9] G. J. Gordon. Stable function approximation in dynamic programming. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, 1995. Also appeared as Technical Report CMU-CS-95-103, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA.

[10] G. J. Gordon. Stable fitted reinforcement learning. In D. Touretzky, M. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, Cambridge, MA, 1996. MIT Press.

[11] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. Technical Report 9307, Department of Brain and Cognitive Sciences, Massachusetts Institut of Technology, July 1993.

[12] L. P. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In P. E. Utgoff, editor, *Proceedings of the Tenth International Conference on Machine Learning*, pages 167–173, San Mateo, CA, 1993. Morgan Kaufmann.

[13] L. P. Kaelbling, M. L. Littman, and A. W. Moore. An introduction to reinforcement learning. In L. Steels, editor, *The Biology and Technology of Intelligent Autonomous Agents*, pages 90–127, Berlin, Heidelberg, March 1995. Springer Publishers.

[14] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8, 1992.

[15] A. Mahdavi. SEMPER: A computational tool for active and multi-domain design and evaluation support. Technical Report 95-04-01, CBPD, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA, 1995.

[16] A.K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Rochester, N.Y., December 1995.

[17] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[18] A. W. Moore. Fast, robust adaptive control by learning only forward models. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 571–578, San Mateo, CA, 1992. Morgan Kaufmann.

[19] A.W. Moore and C.G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21, 1995.

[20] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*, Cambridge, MA, 1998. MIT Press.

[21] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[22] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[23] M.B. Ring. CHILD: A first step towards continual learning. In S. Thrun and L.Y. Pratt, editors, *Learning To Learn*. Kluwer Academic Publishers, Boston, MA, 1998.

[24] S. P. Singh. Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning*, 8, 1992.

[25] S.P. Singh and D. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems 9*, Cambridge, MA, 1997. MIT Press.

[26] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 1988.

[27] R. S. Sutton. Integrated modeling and control based on reinforcement learning and dynamic programming. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 471–478, San Mateo, 1991. Morgan Kaufmann.

[28] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[29] G. J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8, 1992.

[30] S. Thrun. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. Kluwer Academic Publishers, Boston, MA, 1996.

[31] S. Thrun and L.Y. Pratt, editors. *Learning To Learn*. Kluwer Academic Publishers, Boston, MA, 1998.

[32] S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, Cambridge, MA, 1995. MIT Press.

[33] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.

[34] W. Zhang and T.G. Dietterich. High-performance job-shop scheduling with a time-delay td(lambda) network. In D. Touretzky, M. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1024–1030, Cambridge, MA, 1996. MIT Press.